

Lesson 6

Understanding Databases

Learning Objective

Students will learn about:

- Relational Database Management Systems
- Database Query Methods
- Database Connection Methods

Understanding Databases

A database is an organized collection of interrelated data that is managed as a single unit.

A database allows you to store, maintain, and retrieve important data. If a database is properly designed, it can be used by multiple applications and by multiple users. A database management system (DBMS), on the other hand, is software that organizes databases and provides features such as storage, data access, security, backup, etc. Examples of popular DBMSs include Microsoft SQL Server, Microsoft Access, Oracle, and MySQL.

Database management systems can be implemented based on different models. Of these models, the relational model is most popular. In the relational model, data is organized into tables, each of which can have multiple rows. DBMSs based on relational models are called relational DBMSs (RDBMSs). SQL Server, Access, Oracle, and MySQL are all RDBMSs.

Other database management systems are based on different models. For example, object DBMSs (ODBMSs) are based on the object model, in which data is stored as a collection of objects. In this lesson, however, we will focus solely on the more popular relational databases.

Relational DBMSs use Structured Query Language (SQL) to retrieve and manipulate data. Most popular relational database management systems provide some support for the standardized version of SQL, thereby allowing you to use your skills across different relational database systems.

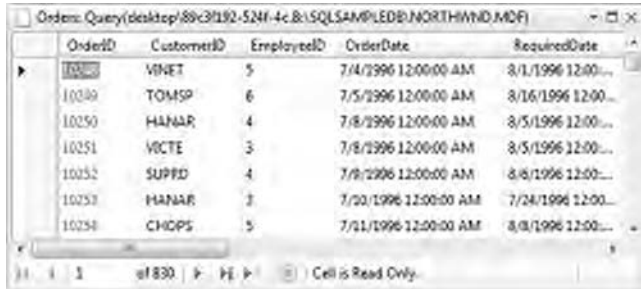
Understanding Relational Database Concepts

A relational database organizes data in two-dimensional tables consisting of columns and rows.

A relational database organizes information into tables. A table is a list of rows and columns that is conceptually similar to a Microsoft Excel worksheet. A row is also called a record or tuple, and a column is sometimes called a field. The column or field specifies the type of data that will be stored for each record in the table. For example, customer orders can be stored in an Orders table in which each row represents a unique order. In this table, columns such as OrderDate can be used to specify that a valid value is of the correct data type. A sample Orders table is shown in Figure 6-1.

Figure6-1

Lesson6: Understanding Databases



The screenshot shows a Microsoft Access query window titled 'Orders: Query (desktop/89c3192-524f-4c8-15QLSAMPLEDB\NORTHWND.MDF)'. The query displays a table with the following columns: OrderID, CustomerID, EmployeeID, OrderDate, and RequiredDate. The data is as follows:

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
10340	VINET	5	7/4/1996 12:00:00 AM	8/1/1996 12:00:00 AM
10349	TOMSP	6	7/5/1996 12:00:00 AM	8/16/1996 12:00:00 AM
10350	HANAR	4	7/8/1996 12:00:00 AM	8/5/1996 12:00:00 AM
10351	VICTE	3	7/8/1996 12:00:00 AM	8/5/1996 12:00:00 AM
10352	SUPRD	4	7/9/1996 12:00:00 AM	8/8/1996 12:00:00 AM
10353	HANAR	3	7/20/1996 12:00:00 AM	7/24/1996 12:00:00 AM
10354	CHOPS	5	7/21/1996 12:00:00 AM	8/9/1996 12:00:00 AM

OBJECTIVE1: Understanding Relational Database Design

Relational database design is the process of determining the appropriate relational database structure to satisfy business requirements.

An organization's data is one of its most important assets. Thus, when you design a database, one of the guiding principles is to ensure database integrity. Integrity means that the data in the database is accurate and consistent at all times.

The database design process consists of the following steps:

1. **Develop a mission statement for the database:** Identifies the purpose of the database, how it will be used, and who will use it. This step sets the tone for the rest of the design process.
2. **Determine the data that needs to be stored:** Identifies all the different types of data that need to be stored in the database. Generally, this information is collected as part of the requirements analysis task via entity-relationship diagrams.
3. **Divide the data into tables and columns:** Identifies the tables and the information that you want to store in those tables.
4. **Choose primary keys:** A primary key is a column or set of columns that uniquely identifies each row of data in a table.
5. **Identify relationships:** Identifies how the data in one table is related to the data in another table. For example, for each customer in a Customers table, you may have many orders in the Orders table; this relationship is called a one-to-many relationship.
6. **Apply the normalization process:** Applies data normalization rules to ensure that any problems that may affect data integrity are resolved. You'll learn more about the normalization process later in this lesson.

After you've established the purpose of a database, the next set of steps (Step 2 through Step 5) can be completed as part of entity-relationship modeling. The final step of normalization can then be applied to the output from this modeling.

Understanding Entity-Relationship Diagrams

Entity-relationship diagrams (ERDs) are used to model entities, their attributes, and the relationships among entities. Entity-relationship diagrams can help you determine what data needs to be stored in a database.

Entity-relationship modeling is a process used to create the conceptual data model of a system, and entity-relationship diagrams are the graphical modeling tools for accomplishing this modeling. The basic building blocks of an ERD are entity, attribute, and relationship:

- **Entity:** An entity is a construct for a physical object or a concept. Examples include an order, a customer, an employee, and so on. An entity is generally named for the noun that it represents.
- **Attribute:** Attributes are the distinct properties of an entity. For example, for an Order entity, some useful attributes may be OrderNumber, OrderDate, ShipDate, and ShipVia. Similarly, for an Employee entity, some useful attributes may be EmployeeId, LastName, FirstName, Title, and HireDate. Every entity must have a set of uniquely identifying attributes that is known as the entity's primary key. For example, an OrderNumber is an attribute that uniquely identifies an order, so it is therefore a primary key for the Order entity
- **Relationship:** A relationship is an association between entities. For example, Takes is a relationship between an Employee entity and an Order entity (i.e., Employee Takes Order).

Note that ERDs don't show single entities or single relations. For example, there may be thousands of Order entities and hundreds of Customer entities. Instead, these diagrams show entity sets and relationship sets—for instance, all the thousands of Order entities may make up one entity set. In fact, when an Order or Customer appears in an ERD, it usually refers to an entity set rather than an individual entity.

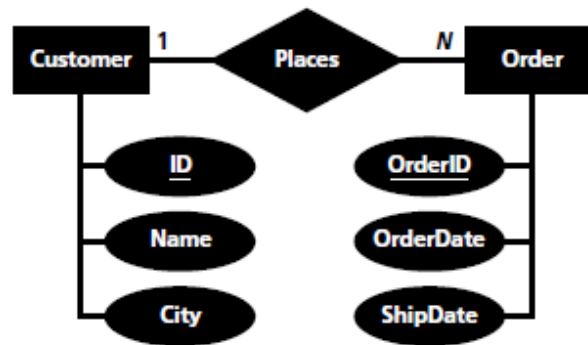
ERDs use certain design conventions. In particular:

- A rectangle represents an entity set.
- An ellipse represents an attribute.
- A diamond represents a relationship set.
- Solid lines link entity sets to relationship sets and entity sets to attributes.

Figure 6-2 shows an example ERD. In this diagram, the two entity sets are Customer and Order. Attributes associated with Customer are ID, Name, and City. Attributes associated with Order are OrderID, OrderDate, and ShipDate. Those attributes that form a primary key are underlined. Also, as shown in the figure, the relationship between Customer and Order is Places.

Figure 6-2

An entity-relationship diagram



Within an ERD, a relationship can be classified as a one-to-one relationship, a one-to-many relationship, or a many-to-many relationship. In Figure 6-2, the line that connects the relationship Places with the entity set Customer is labeled “1,” whereas the line between Places and the entity set Order is labeled “N.” This is an example of a one-to-many relationship. In this relationship, one customer can place many orders, but an order can have only one customer associated with it.

MAPPING ERDs TO A RELATIONAL DATABASE

In order to convert an ERD to a relational database, you must take following steps:

1. **Map the entities:** Start by creating a table for each entity set in the diagram. The attributes will become columns. Be sure to set the primary key attribute(s) to the primary key column(s) for the table.
2. **Map the relationship:** Next, map the one-to-many relationship by ensuring that the table on the N side of the relationship contains the primary key column of the table on the 1 side of the relationship. For Figure 6-2, this can be accomplished by adding a CustomerID column in the Order table that maps to the ID column of the Customer table. In the context of the Order table, the CustomerID is also called a foreign key. By adding this column in the Order table, it is possible to answer questions such as “What are all the orders placed by a specific customer?” and “Who is the customer for a specific order?”

When mapped to a relational database, the ERD in Figure 6-2 generates the following tables:

Customers

ID	NAME	CITY
1001	Jane Doe	Berlin
1002	John Doe	Tokyo
1003	Howard Steel	Sydney

Orders

ORDERID	CUSTOMERID	ORDERDATE	SHIPDATE
101	1001	10/1/2010	10/7/2010
102	1002	10/5/2010	10/10/2010
103	1001	10/4/2010	10/10/2010

Understanding Data Normalization

The process of data normalization ensures that a database design is free of any problems that could lead to loss of data integrity.

Entity-relationship analysis helps you ensure that you've identified the correct data items for your database. Then, through the process of data normalization, you apply a set of normalization rules to make sure that you have established the correct database design—that is, you check whether the columns belong to the right tables in order to ensure that your database is free of any undesirable problems.

For example, as part of entity-relationship analysis, you may come up with a Books table that has the following columns:

Books

BookId	BookName	CATEGORYID	CATEGORYNAME
1	Cooking Light	1001	Cooking
2	Prophecy	1002	Mystery & Thriller
3	Shift	1003	Business
4	The Confession	1002	Mystery & Thriller

However, this design for the Books table suffers from three problems:

- **Insert anomaly:** An insert anomaly is a situation in which you cannot insert new data into a database because of an unrelated dependency. For example, if you want your database to have a

Lesson6: Understanding Databases

new CategoryId and CategoryName for history books, the current design will not permit that unless you first have a history book to place in that category.

- **Delete anomaly:** A delete anomaly is a situation in which the deletion of one piece of data causes unintended loss of other data. For example, if you were to delete the BookId 3 from the Books table, the very fact that you ever had a CategoryName of Business would be lost.
- **Update anomaly:** An update anomaly is a situation in which updating a single data value requires multiple rows to be updated. For example, say you decide to change the Mystery & Thriller category name to just Mystery. With the current table design, you'll have to change the category name for every book in that category. There is also a risk that if you update the category name in one row, but not the others, you'll end up having inconsistent data in the database.

Each of these problems can be fixed by following the normalization process. There are five normal forms that are used as part of this process; however, this lesson only discusses the first three, because they are all that is required in most cases.

Take Note

Normalization can help you ensure a correct database design, but it cannot ensure that you have the correct data items to begin with.

UNDERSTANDING THE FIRST NORMAL FORM

In order for a table to be in the first normal form (1NF), none of the columns in the table should have multiple values in the same row of data. For example, if a Customers table stores data as shown below, this table is not in 1NF because the PhoneNumber column is storing more than one value in each row

Take Note

A general convention is to underline the name of the columns in a table that are part of the primary key.

Lesson6: Understanding Databases

Customer

Id	FIRSTNAME	LASTNAME	PHONENUMBER
1	Jane	Doe	(503) 555-6874
2	John	Doe	(509) 555-7969, (509) 555-7970
3	Howard	Steel	(604) 555-3392, (604) 555-3393

For this table to be in 1NF, you would need to break the table in two:

Customer

Id	FIRSTNAME	LASTNAME
1	Jane	Doe
2	John	Doe
3	Howard	Steel

CustomerPhones

Id	PHONENUMBER
1	(503) 555-6874
2	(509) 555-7969
2	(509) 555-7970
3	(604) 555-3392
3	(604) 555-3393

Here, the Customers table and the CustomerPhones table are both in 1NF. Both tables have a primary key (Id in the first table and the combination of Id and PhoneNumber in the second table) that establishes a relationship between them. Given any Id for a customer, you can find all phone numbers for that customer without any confusion. On the other hand, LastName is not a primary key because a last name may have duplicate entries.

TakeNote

Creating repeating columns such as PhoneNumber1 and PhoneNumber2 to normalize the Customer table would not be an acceptable solution because the first normalization form does not allow such repeating columns.

UNDERSTANDING THE SECOND NORMAL FORM

Lesson6: Understanding Databases

For a table to be in second normal form (2NF), it must first meet the requirements for 1NF. In addition, 2NF requires that all non-key columns are functionally dependent on the entire primary key.

In order to understand 2NF, you must first understand functional dependence. Let's take the example of the Customers table above. In the Customers table, the Id column is the primary key because it uniquely identifies each row. The columns FirstName and LastName are non-key columns, because they are not part of the primary key. Both FirstName and LastName are functionally dependent on Id because, given a value of Id, you can always find a value for the corresponding FirstName and LastName without any ambiguity. There is no non-key column in the Customers table that does not functionally depend on the primary key. The Customers and CustomerPhones table are therefore already in 2NF.

TakeNote

2NF only applies to tables that have composite primary keys (i.e., multiple columns together make up the primary key). The combined values of all fields in a composite primary key must be unique. If a table satisfies 1NF and has only a single column in its primary key, then the table also conforms to 2NF.

In contrast, consider the following table

Orders

ORDERID	CUSTOMERID	ORDERDATE	CUSTOMERNAME
101	1	10/1/2010	Jane Doe
102	2	10/5/2010	John Doe
103	1	10/4/2010	Jane Doe

Here, the OrderId and CustomerId columns together identify a unique row and therefore make up a composite primary key. However, the column OrderDate is functionally dependent only on OrderId, and the column CustomerName is dependent only on CustomerId. This violates the 2NF because the non-key columns are functionally dependent on only part of the primary key.

One possible way you could modify the Orders table to conform to 2NF is to take CustomerName out of the table and have only three columns—OrderId, CustomerId, and OrderDate—with only OrderId serving as the primary key. In this solution, both CustomerId and OrderDate are functionally dependent on OrderId and thus conform to 2NF.

UNDERSTANDING THE THIRD NORMAL FORM

The third normal form (3NF) requires that 2NF is met and that there is no functional dependency between non-key attributes. In other words, each non-key attribute should be dependent on only the primary key and nothing else. For example, consider the following table:

Lesson6: Understanding Databases

Items

ITEMID	SUPPLIERID	REORDERFAX
101	100	(514) 555-2955
102	11	(514) 555-9022
103	525	(313) 555-5735

Here, ItemId is the primary key. However, ReorderFax is a fax number for the supplier and is therefore functionally dependent on SupplierId. To satisfy the requirement of 3NF, this table should be decomposed into two tables: Items (ItemId, SupplierId) and Supplier (SupplierId, ReorderFax).

Items

ITEMID	SUPPLIERID
101	100
102	11
103	525

Supplier

SUPPLIERID	REORDERFAX
100	(514) 555-2955
11	(514) 555-9022
525	(313) 555-5735

OBJECTIVE 2: Understanding Database Query Methods

Data is at the core of many business applications, and, as a developer, you will likely spend a lot of time working on data-related tasks. In this section, you will learn how to use Structured Query Language (SQL) and SQL Server-stored procedures to select, insert, update, and delete data

SQL is the language used by most database systems to manage the information in their data-bases. SQL commands permit you to retrieve and update data. SQL commands also let you create and manage database objects such as tables. SQL may be thought of as a programming language for relational databases. However, SQL is declarative in nature, as opposed to the imperative nature of most common programming languages.

In SQL, you tell the database what needs to be done, and it's the database's job to figure out how to do it—for example, you can tell the database to select the first 10 rows from a table. Compare this with an imperative programming language such as C#, in which you need to specify in detail how the work is to be

Lesson6: Understanding Databases

performed. For example, you might need to create a loop that runs ten times, set up and initialize variables, move record pointers, and so on.

SQL is an ANSI (American National Standards Institute) standard, but different database vendors have implemented their own extensions to standard SQL. Microsoft SQL Server's implementation of SQL is called Transact-SQL (T-SQL).

There are two main ways to submit T-SQL to SQL Server. You can either use ad-hoc SQL statements that are executed directly, or you can use stored procedures. Stored procedures are collections of SQL statements and programming logic that are stored on the database server as named objects.

Working with SQL Queries

SELECT, INSERT, UPDATE, and DELETE statements are the four main types of SQL statements used to manipulate SQL Server data

Using adhoc SQL queries is a flexible way to work with a SQL Server database. In this portion of the lesson, you'll learn the basics about the four main types of SQL statements that help you manipulate SQL Server data:

- SELECT statements allow you to retrieve data stored in a database.
- INSERT statements allow you to add new data to a database.
- UPDATE statements allow you to modify existing data in a database.
- DELETE statements allow you to delete data from a database.

CONNECTING TO A SQL SERVER DATABASE

You need to connect to a SQL Server database before you can manipulate any information in that database.

In this exercise, you'll learn how to work with a Microsoft SQL Server database. If you don't have access to a recent version of SQL Server, you can download SQL Server 2008 Express for free from www.microsoft.com/express/database. This exercise uses the SQL Server sample database Northwind. This database is not installed by default with SQL Server, but you can download the database file by following the instructions at www.msdn.com/en-us/library/ms143221.aspx.

Complete the following exercise to connect to and use the Northwind database with Visual Studio.

Illustration 1: CONNECT TO NORTHWIND DATABASE

GET READY. Before you begin these steps, be sure to launch Microsoft Visual Studio.

1. Open the Server Explorer window. Select the Data Connections node, then click the Connect to Database button on the Server Explorer toolbar

TakeNote

In Visual Studio Express Edition, the Server Explorer window is called Database Explorer, and it can be opened by

selecting View > Other Windows > Database Explorer

2. In the **Add Connection** dialog box, browse to the database file for the Northwind database (northwnd.mdf), as shown in Figure 6-3.

Figure 6-3

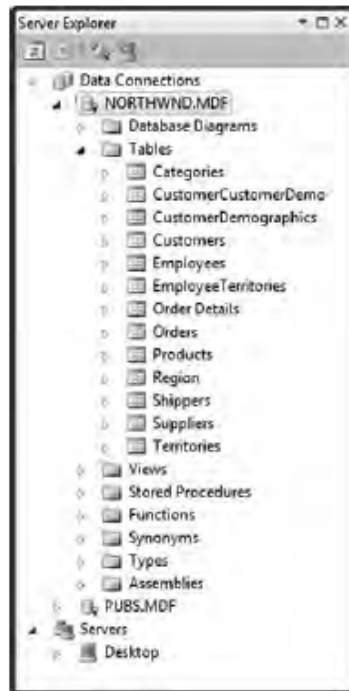
Connecting to the Northwind database



3. Use Windows Authentication as the authentication mode, and click the **Test Connection** button to make sure you can connect to the database. Finally, click the **OK** button to add the connection to the database.
4. Once the connection is established, the database is available as a connection under the Data Connections node in Server Explorer. Expand the database to see the tables, stored procedures, and other database objects, as shown in Figure 6-4

Figure 6-4

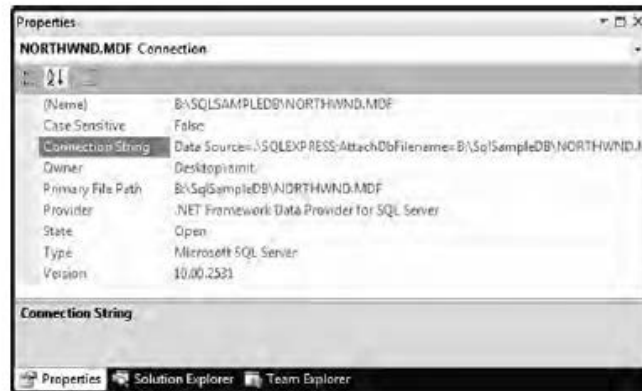
Accessing the Northwind database through Server Explorer



5. Right-click the **NORTHWND.MDF** node and select **Properties**. You should see the Properties window shown in Figure 6-5. In this window, notice the Connection String property. You'll use the value of this property to connect to the Northwind database from a C# application

Figure 6-5

Properties window for the Northwind database



PAUSE. You will access data from the Northwind database in the next exercise

RUNNING SQL QUERIES

There are many ways to communicate with SQL Server in order to run database queries. There are many ways in which you can send queries to a SQL Server. For example, you can use any of the following:

- Visual Studio Integrated Development Environment (IDE)

Lesson6: Understanding Databases

- C# application
- SQL Query Analyzer
- osql command prompt utility

Note that SQL Query Analyzer and osql command prompt utilities are tools installed with SQL Server.

Illustration 2: RUN QUERIES FROM VISUAL STUDIO

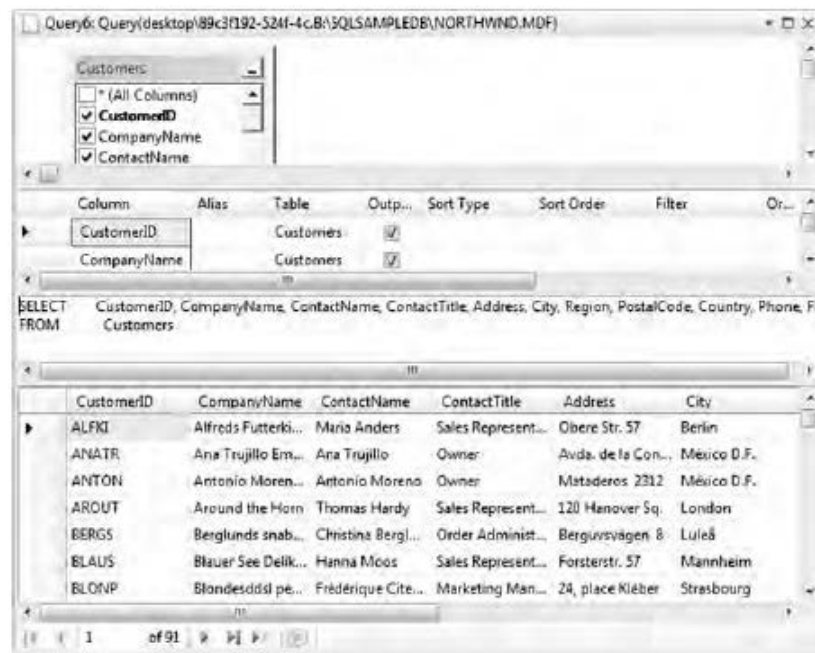
GET READY. To use Visual Studio IDE and C# applications to run SQL queries, perform these steps:

1. Select the Northwind database in the Server Explorer. Right-click the database and select New Query. This action opens a Query Designer and shows an Add Table dialog box. Select the Customers table and click Add. Click Close in the Add Table dialog box.
2. In the SQL pane of the Query Designer (which is the area that displays the text of the query), modify the SQL statement to the following:

```
SELECT * FROM Customers
```

3. From the Visual Studio menu, select the Query Designer > Execute SQL option, or click on the Execute SQL button in the toolbar. The SQL statement will be sent to the SQL server for execution, and results like those in Figure 6-6 should be displayed

Figure 6-6
Visual Studio Query Designer



The Query Designer in Visual Studio displays up to four panes. From top to bottom, the panes are as follows:

- **Diagram pane:** This pane displays the tables involved in the query and the relationships among these tables, as well as all the columns that the tables contain.
- **Criteria pane:** The Criteria pane shows the columns that have been selected as part of the query, as well as additional sorting and filtering information.
- **SQL pane:** This pane shows the actual SQL statement that will be executed.

Lesson6: Understanding Databases

- **Results pane:** The Results pane shows the results (if any) after the query has been executed.

The Query Designer toolbar includes buttons that you can use to hide or show any of these four panes.

For the following exercise, you need only the SQL pane and the Results pane

Illustration 3: RUN QUERIES FROM C# APPLICATION

GET READY. To run queries from C# applications, do the following:

1. Create a new Windows Application project named QueryCS.
2. To the Windows Form, add a TextBox control, a Button control, and a DataGridView control. Set the MultiLine property of the TextBox to True. Set the Text property of the Button control to Execute SQL.
3. Double-click the Button control to generate an event handler for its Click event. Modify the event handler as shown below:

```
private void button1_Click (object sender, EventArgs e)
{
    if (textBox1.TextLength > 0)
    {
        SelectData(textBox1.Text);
    }
}
```

4. Add the following method to the class. Be sure to change the connection string to match the local path of the database file on your computer:

```
private void SelectData(string selectCommandText)
{
    try
    {
        // Change the connection string
        // to match with your system. string selectConnection =
        @"Data Source=.\\SQLEXPRESS;" +
        @"AttachDbFilename=" +
        @"c:\\SqlSampleDB\\NORTHWND.MDF;" +
        @"Integrated Security=True;" +
        @"Connect Timeout=30;User Instance=True";

        SqlDataAdapter dataAdapter = new SqlDataAdapter( selectCommandText, selectConnection);
        DataTable table = new DataTable();
        dataAdapter.Fill(table);
        dataGridView1.DataSource = table
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

}

5. Add the following using directives to the code:

using System.Data;

using System.Data.SqlClient;

Select Debug > Start Debugging to run the project. Enter a valid SQL query and click on the Button control. You should see the output shown in Figure 6-7.

Figure 6-7

Running queries from a C# application



The code in this exercise implements a `SelectData` method that initializes a `SqlDataAdapter` object and uses it to populate a `DataTable`. The `DataTable` is then bound as a data source for the `DataGridView` component. The `SqlDataAdapter` object acts as a pipeline between SQL Server and the `DataTable` for retrieving data. The `Fill` method changes the data in the `DataTable` to match the data in the data source. The `selectCommandText` is used to identify the data in the data source

SELECTING DATA

The `SELECT` statement is used to retrieve data from one or more database tables.

The `SELECT` statement generally takes the following form:

```
SELECT list_of_fields
FROM list_of_tables
WHERE where_clause
GROUP BY group_by_clause
HAVING having_clause
ORDER BY order_by_clause
```

Each of these lines of code in the `SELECT` statement is called a clause. The `SELECT` and `FROM` clauses are required, but the rest are optional. For example, here's a SQL statement that contains only the required clauses:

SELECT OrderId, CustomerId

FROM Orders

If you want to list all the fields from a table, you can also use the following shortcut instead of explicitly listing all the fields:

SELECT *

FROM Orders

In addition, you can select information from multiple tables; for example:

**Select OrderId, Customers.CustomerId, ContactName
From Orders, Customers**

Customers.CustomerId is known as a fully qualified name because it specifies both the table name and field name. This is necessary because both the Orders table and the Customers table include this field, so you must tell SQL Server which particular table you want to refer to.

If you run this query, you will get a lot more records than you might expect. This happens because, although you told SQL Server what tables to include, you didn't include any information on how to relate those tables. As a result, SQL Server constructs the result set to include all rows of the Customer table for every row of the Orders table. This kind of join is called a cross join, and it is not very helpful in this case.

A more useful query, of course, would match each order with the corresponding customer.

The INNER JOIN keyword can help you accomplish this, as shown in the following query:

**SELECT OrderID, Customers.CustomerId, ContactName
FROM Orders INNER JOIN Customers
ON Orders.CustomerId = Customers.CustomerId**

This query tells SQL Server to take each row in the Orders table and match it with all rows in the Customers table in which the CustomerId of the order equals the CustomerId of the customer. Because CustomerId is unique in the Customers table, this is the same as including only a single row for each order in the result set. In this case, the result set will have as many rows as there are rows in the Orders table.

But what if you want to see only some of the rows in the table? In this situation, you can use the WHERE clause. The WHERE clause evaluates each row for a condition and decides whether to include it in the result set. For example:

**SELECT * FROM Orders
WHERE ShipCountry = 'Canada'**

Here, the WHERE clause looks at every row in the Orders table to see whether the ShipCountry has the exact value "Canada." If it does, the row is included in the result set; if it does not, the row is not included in the result set.

You can also combine multiple conditions in a single WHERE clause. For example:


```
SELECT * FROM Orders  
WHERE (ShipCountry = 'Canada')  
AND (OrderDate >= '01/01/97')  
AND (OrderDate <= '01/31/97')
```

Here, the WHERE conditions filters the orders in which the ShipCountry is “Canada” and the order date is in January 1997.

By default, SQL does not guarantee the results to be in a particular order. However, you can use the ORDER BY clause to ensure that your desired data is returned in a particular order. For example, to list the orders based on their order date, you can use the following query:

```
SELECT * FROM Orders  
WHERE (ShipCountry = 'Canada')  
AND (OrderDate >= '01/01/97')  
AND (OrderDate <= '01/31/97')  
ORDER BY OrderDate
```

You can modify the sort order by using either the keyword ASC (for ascending order) or the keyword DESC (for descending order). The default sort order is ascending. Thus, the following query lists the most recent orders at the top:

```
SELECT * FROM Orders  
WHERE (ShipCountry = 'Canada')  
AND (OrderDate >= '01/01/97')  
AND (OrderDate <= '01/31/97')  
ORDER BY OrderDate DESC
```